1-1-2006

Improving object cache performance through selective placement

Saied Hosseini-Khayat
University of Wollongong

Improving object cache performance through selective placement

Abstract

Distributed systems greatly benefit from caching. Caching data objects of variable size and cost poses interesting questions that have been researched for the past ten years. As a result, a few good algorithms have come to the fore. These algorithms make effective decisions in selecting cache objects for removal. However, they make no decision about the suitability of a new object for placement into the cache. We show that “selective placement” can add further improvement to these algorithms when a request pattern consists of frequent references to a working set of objects interspersed with isolated references to less popular objects. The key idea is to avoid indiscriminate caching, and to weigh the benefits of caching an object against the cost of removing other objects. This paper describes a simple enhancement to a well-known web caching algorithm (GreedyDual-Size) to make it a selective algorithm. It is shown by simulation that the performance gain can be substantial. The suggested methodology can be applied to similar algorithms.

Keywords

Improving, object, cache, performance, through, selective, placement

Disciplines

Physical Sciences and Mathematics

Publication Details

Hosseini-Khayat, S. 2006, "Improving object cache performance through selective placement", in T. Fahringer (eds), International Conference on Parallel and Distributed Computing and Networks, ACTA Press, Anaheim, USA, pp. 262-265.

IMPROVING OBJECT CACHE PERFORMANCE THROUGH SELECTIVE PLACEMENT

Saied Hosseini-Khayat
University of Wollongong, Dubai Campus
Knowledge Village, Dubai, UAE
saiedk@uow.edu.au

ABSTRACT
Distributed systems greatly benefit from caching. Caching data objects of variable size and cost poses interesting questions that have been researched for the past ten years. As a result, a few good algorithms have come to the fore. These algorithms make effective decisions in selecting cache objects for removal. However, they make no decision about the suitability of a new object for placement into the cache. We show that "selective placement" can add further improvement to these algorithms when a request pattern consists of frequent references to a working set of objects interspersed with isolated references to less popular objects. The key idea is to avoid indiscriminate caching, and to weigh the benefits of caching an object against the cost of removing other objects. This paper describes a simple enhancement to a well-known web caching algorithm (GreedyDual-Size) to make it a selective algorithm. It is shown by simulation that the performance gain can be substantial. The suggested methodology can be applied to similar algorithms.

KEY WORDS
Selective cache, cache replacement, web caching.

1 Introduction

Caching is perhaps the single most effective technique to improve the scalability and usability of distributed systems. A variety of large scale distributed systems are in use today (such as the World Wide Web) and are envisioned to appear in the future (such as content delivery networks). The use of effective caching strategies is necessary in those systems because they can reduce both network traffic and user-perceived latency. In this context, the idea of object caching is to store the most frequently requested data object (text, images, audio, video) in servers that are close to the point of request. This is the idea behind "web caching proxies" or "proxy servers" in the World Wide Web.

In an object cache, as opposed to a CPU cache, the size and the cost of data objects are variable. *Size* is normally the size of objects measured in bytes, and *cost* quantifies the amount of network load or user-perceived latency (depending on the optimization goal) caused by different objects. The performance of an object cache is measured in terms of the average accumulated miss cost over a long sequence of cache requests.

One of the main factors affecting cache performance is the replacement policy that is used to replace stale objects when new requests arrive. Since the rise of the World Wide Web, a great deal of research has gone into such replacement policies. In general, the algorithms that combine size, cost, and activeness in a meaningful ways have been shown to do an effective job in web caching. For a recent survey of replacement algorithms refer to [2]. Also for the most recent bibliographies of web caching refer to [3] and [4]. The successful algorithms typically tend to replace those objects that have large size, small cost and are least frequently requested. To make effective use of the limited cache space, the above criteria is reasonable. However, it is also reasonable to question whether every new item has to be placed into the cache. It is quite possible that a newly requested object is not valuable enough to be placed into the cache, especially if caching this item causes valuable object(s) to be removed. This is a particularly bad decision when the new item is large but not a frequently requested one.

In this paper, we propose a selective placement strategy that avoids bad placement decisions. To show that the proposed strategy works, we use a well-known web caching algorithm, GreedyDual-Size, as a benchmark and enhance it with our proposed placement scheme. Then we measure the performance of the two algorithms and make comparison.

In the next section, we explain the key idea of this paper. In Section 3, a new caching algorithm which is based on the key idea is introduced. In Section 4, the proposed algorithm is evaluated by simulation. The conclusion of paper follows in Section 5 in which we summarize the contribution and suggest future work.

2 Key Idea

It is well known that a sizable portion of web requests as seen by proxy servers consists of isolated non-repeating requests ([1],[5]). There are two reasons cited for this: (1) the browser caches absorb a lot of short-term repeats, (2) frequency of object requests follow a Zipf-like distribution. (In a Zipf-like distribution, the frequency of requested items in a long time interval drops inversely proportional to its "popularity" rank.) The same statement is

potentially true for other distributed services because at any given time, only a small portion of objects are "popular." For instance, in a video distribution network, only a small subset of movies are in popular demand on any day, and there are infrequent isolated requests for outdated movies as well. This request behavior is said to follow a "working set model."

The key observation of this paper is the following: Caching the popular objects can result in performance gain, whereas caching the isolated requested objects not only does not contribute positively to performance but also degrades it. This observation leads to the following guidelines:

(a) Not every requested object should be necessarily cached. A *placement policy* must be in place to decide whether to cache an object.

(b) The placement policy should weigh the potential cost of not caching a new object against the potential cost of removing one or more cached objects.

(c) The placement policy should estimate the "popularity" of an item and use this information when performing task (b).

(d) The placement policy can be independent of a replacement policy which will be applied afterwards. The two policies may as well be combined into an integrated placement/replacement policy.

The term "selective caching" is not new and has been used in CPU caching [7]. However in most (if not all) of the web replacement algorithms (see [2]), it is taken for granted that every requested object must be placed into cache (contrast this with item (a) above). To do (c), there is no unique way to estimate "popularity" of requested objects. One way is to use the frequency of requested objects in a time interval, as we use in this paper.

The placement and replacement policies do not have to be separate independent procedures. In fact, many replacement algorithms can be modified to perform selective placement and replacement at the same time. Most replacement algorithms perform removal by assigning a "value" to objects in the cache. Then they remove objects having the least values. Instead of assigning a value to objects residing in cache, they may as well assign a value to the missing newly-requested object. If the object gets the least value, it will not be placed in the cache.

In this paper, we propose the following selective caching methodology. This methodology can produce a variety of caching algorithms by filling it in with specific placement and replacement policies.

1. For every requested object p, if p exists in the cache, then abort this procedure.

2. If p is not in cache but *size*$(P) \leq$ *FreeSpace*, then place p into cache.

3. If p is not in cache and *size*$(P) >$ *FreeSpace*, then use the placement policy. If this policy determines that p should not be placed into cache, then abort this procedure. Otherwise do the following step.

4. Use replacement policy to decide which objects to remove. Remove those objects and then insert p into cache.

In the following sections, this idea is explored by modifying a known caching algorithm.

3 Proposed Algorithm

Among the numerous algorithms proposed for web caching, there is no algorithm with a decisive performance advantage in all situations. However, a few have stood out as "good enough" algorithms [2]. One such algorithm, introduced by Cao and Irani [6], is called *GreedyDual-Size*. This algorithm is a competitive online algorithm meaning that its online cost is guaranteed to be within a constant factor of an optimal offline algorithm [6].

In this paper, we choose the *GreedyDual-Size* replacement algorithm and enhance it with our placement policy *Window-based Frequency Estimation* (WFE) to make a selective caching algorithms. The idea behind WFE placement is to estimate the "popularity" of an object by running a sliding window on the past requests.

Consider an ongoing sequence of requests, and suppose we are dealing with the current request which is the j^{th} request. The size of our sliding window is a fixed constant W and start from the $(j-1)^{th}$ request backwards to the $(j-W-1)^{th}$ request. (This window is implemented as a circular array of object identifiers.) We say an object P has an estimated frequency $\hat{f}(p) = k$ if p is requested k times in the window. ($\hat{f}(p) = 0$ if p is not requested in the span of the window.) The frequency of objects existing in the window gives an estimate of the actual popularity of objects. Note that popularity can vary with time, therefore having a limited sliding window offers adaptability to changes of popularity.

Now we present our proposed adaptation of the *GreedyDual-Size* algorithm which we call *Selective GreedyDual-Size*. In this algorithm (Figure 1), $c(p)$, $s(p)$ and $\hat{f}(p)$ denote the cost, size and estimated frequency of object p. The placement policy in the above algorithm is given by the condition $\hat{f}(p) \geq T$, where T is a constant threshold parameter. It permits placement of an object into cache if its estimated frequency is above a given threshold value. The estimated frequencies are constantly updated as the window slides forward. (This update is efficiently done by incrementing the frequency of the object going into the window's scope and decrementing that of the object going out of scope.) This algorithm has two parameters W and T to be tuned for optimal performance. Their optimal values depend on the request sequence at hand. Note that the original *GreedyDual-Size* algorithm is obtained by setting

Selective GreedyDual-Size
Given: Cache S, Request sequence R
Initialize $L \leftarrow 0$
For each p in R in turn do
  If p is in cache S then
    $H(p) \leftarrow L + c(p)/s(p)$
  Else if $\hat{f}(p) \geq T$ then
    While $FreeSpace(S) < s(p)$ do
      Let $L \leftarrow \min_{q \in S} H(q)$
      Evict q such that $H(q) = L$
    Load p into cache
    $H(p) \leftarrow L + c(p)/s(p)$
  Slide window one step forward and
  Update estimated frequencies.

Figure 1. Proposed algorithm

$T = 0$. In the next section, we evaluate the performance of the above algorithm through simulation.

4 Performance Evaluation

To evaluate and compare the performance of *Selective GreedyDual-Size* algorithm with that of the original algorithm, we used the following simulation settings:

- A set of 10,000 cache objects were considered. The size of these objects where generated according to two discrete distributions, in one experiment Geometric (negative exponential-shaped) and in another experiment Binomial (bell-shaped). The cache size was always chosen to be greater than the maximum size. The cost of objects were also random but strongly and positively correlated with their sizes. The rational is that large objects cause more network traffic and longer user-perceived delay than the small object.
- An independently and identically distributed sequence of 1000,000 requests were generated according to a Zipf-like distribution. Among the 10,000 objects, roughly 5% of them were requested 45% of the time, and 1% of them were requested 30% of the time. The rest of the times the remaining objects were requested with uniform likelihood. The idea is to simulate a large set of cacheable objects only a small portion of which is highly popular at any time.
- The measure of performance of interest is the **miss cost ratio** which is *the accumulated cost of missing objects divided by the total cost of all requests* during the interval of observation.

Two experiments were carried out which differed only in the distribution of object sizes. The first used a bell-shaped distribution with a mean around 10 KBytes. The second used a negative exponential-shaped distribution with a mean around 20 KBytes. In both experiment, the cache size was varied from 1/2 MBytes to 7 MBytes. Also the threshold parameter T and Window size W were varied in separate simulation runs. The resulting performance graphs are shown in Figures (2), (3), (4), and (5).

Figure 2. Binomial size distribution (W=1000)

Figure 3. Binomial size distribution (W=2000)

In all four figures, the particular graph corresponding to $T = 0$ shows the miss cost ratio of *GreedyDual-Size* algorithm (the benchmark), while the other three graphs in each figure corresponding to $T = 1, 2, 3$ show the miss cost ratio for *Selective GreedyDual-Size* with different threshold values. In our experiments, *Selective GreedyDual- Size* turned out to perform significantly better than the original algorithm. The superiority tends to fade away at large cache sizes, which is reasonable. The right threshold T for the placement policy seems to be 2 or 3. Larger window

Figure 4. Geometric size distribution (W=1000)

Figure 5. Geometric size distribution (W=2000)

sizes also take away some performance advantage from our placement scheme. There is a best value for T and W which has to be figured out for a given situation.

5 Conclusion

In this paper, it was argued that if the request pattern for a given distributed system follows the "working set model" then it is possible to improve object cache performance by selective caching. In selective caching for any new request the algorithm should decide whether the response should be cached. In the case of requests to isolated non-popular items, it seems it would be better to avoid caching which normally requires eviction of some potentially valuable objects. It was shown by means of simulations that a well-known web caching algorithm, *GreedyDual-Size*, can be significantly enhanced by adding a selective placement policy. The placement policy works by estimating the popularity of objects and allowing the placement of those objects which have an estimated frequency larger than a fixed threshold value. Estimation of frequencies is performed by means of a sliding window over the past requests. The length of this window and the threshold value are tuning parameters and can have considerable impact on performance. Finding adaptive ways to set these values is an interesting topic for future research. The suggested performance improvement is not specific to *GreedyDual-Size* and can be applied to other object caching algorithms as well. A more extensive study of selective strategies including trace-driven simulations is left for future research.

References

[1] Michael Rabinovich, Oliver Spatscheck, *Web Caching and Replication*, 2002, Addison-Wesley, Boston, MA.

[2] Stefan Podlipnig, Laszlo Boszormenyi, "A Survey of Web Cache Replacement Strategies," *ACM Computing Surveys*, Vol. 35, No. 4, December 2003, pp. 374–398.

[3] "Web Caching Bibliography" currently available online at *http://www-sor.inria.fr/projects/relais/biblio/webcaches.bib*.

[4] Brian M. Davison, "Brian Davison's Web-Caching Bibliography," currently available online at *http://www.web-caching.com/*, January 2001.

[5] Guangwei Bai, Carey Williamson, "Workload Characterization in Web Caching Hierarchies," Proceedings of the 10th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems, 2002.

[6] Pei Cao, Sandy Irani, "Cost-Aware WWW Proxy Caching Algorithms," *Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems*, pp.193–206, December 1997.

[7] L. John, R. Radhakrishnan, "A Selective Caching Technique," currently available online at *http://citeseer.ist.psu.edu/115017.html*.